REPORT DOCUMENTATION PAGE

1

Form Approved OMB NO. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (From - To)
29-08-2014	Conference Proceeding	-

4. TITLE AND SUBTITLE
Validation of 3D RANS-SA Calculations on Strand/Cartesian Meshes

5a. CONTRACT NUMBER
W911NF-12-1-0008

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
622307

6. AUTHORS
Andrew M. Wissink, Nicholas K. Burgess, Aaron J. Katz, Jayanarayanan Sitaraman, Robert Haimes

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES
Utah State University
1415 Old Main Hill - Room 64

Logan, UT 84322 -1415

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES)
U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
60390-EG.28

12. DISTRIBUTION AVAILIBILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT
This paper will present validations of the 3D RANS-SA solver presented in earlier work? to automatically-generated strand meshes. Included will be high-Re calculations performed with Strand/Cartesian mesh systems on practical 3D geometrically-complex applications used for engineering analysis.

15. SUBJECT TERMS
RANS, strand grids

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	15. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Aaron Katz
UU	UU	UU	UU		19b. TELEPHONE NUMBER 435-797-7021

Report Title

Validation of 3D RANS-SA Calculations on Strand/Cartesian Meshes

ABSTRACT

This paper will present validations of the 3D RANS-SA solver presented in earlier work? to automatically-generated strand meshes. Included will be high-Re calculations performed with Strand/Cartesian mesh systems on practical 3D geometrically-complex applications used for engineering analysis.

Conference Name: AIAA SciTech 2014

Conference Date: January 07, 2014

21st AIAA Computational Fluid Dynamics Conference, June 24-27, 2013, San Diego, CA

Validation of 3D RANS-SA Calculations on Strand/Cartesian Meshes

Andrew M. Wissink [*], Nicholas K. Burgess [†]
and
Aaron J. Katz [‡], Jayanarayanan Sitaraman [§],
and
Robert Haimes [¶]

This paper will present validations of the 3D RANS-SA solver presented in earlier work[7] to automatically-generated strand meshes. Included will be high-Re calculations performed with Strand/Cartesian mesh systems on practical 3D geometrically-complex applications used for engineering analysis.

I. Introduction

As computational fluid dynamics (CFD) becomes a more integral part of the engineering design process improvements are necessary in both automation and computational efficiency. Adaptive Cartesian Euler solvers[1–4] have currently achieved a degree of automation that they are now used regularly by design engineers for analysis of new conceptual designs. The availability of automated adjoint-based mesh refinement[5] also permits their use in design optimization.[6]

The same cannot be said of viscous-based Reynolds Averaged Navier Stokes (RANS) and Detached Eddy Simulation (DES) solvers. Viscous mesh generation continues to be a manually-driven process, often requiring expert analysts multiple days to weeks to construct a high quality mesh around geometrically complex vehicles. Additionally, when mesh generation and flow solution are treated as separate non-synergistic operations it imposes a barrier to improving solution accuracy. That is, the quality of the solution is dependent on the quality and resolution of the underlying mesh. If the mesh is fixed the solver has no mechanism to improve solution quality through mesh manipulation or refinement.

The overset dual-mesh "strand"–Cartesian approach has been proposed and studied in earlier works[7–13] as a viable means to support automatic viscous mesh generation and adaptation. In the strand paradigm, a body-fitted near-body mesh is constructed by a set of straight line segments grown directly from the surface, each with the same point distribution in the normal direction, forming a thin layer of mostly prismatic elements around the body. Once outside the viscous boundary layer, strands transition to isotropic block structured Cartesian grids. The two grid systems intersect through overlapping chimera overset procedures. The procedure is similar in concept to standard prismatic unstructured grid generation techniques, in which prismatic cells are grown at the surface in the viscous boundary layer with tetrahedra elsewhere, except in the strand approach Cartesian grids are used in place of tetrahedra for the Euler solution. Figure 1 shows an example calculation using the strand-Cartesian paradigm.[8]

In addition to streamlined and automatic meshing capability, the strand-Cartesian approach presents three other important advantages. First, both strand and Cartesian meshes may be represented with extremely low memory descriptions, enabling the entire global mesh description to fit on each processor in

[*]Aerospace Engineer, U.S. Army Aerodynamics Development Directorate (AMRDEC), Moffett Field CA, AIAA Member
[†]Aerospace Engineer, STC Corp, NASA Ames Research Center, Moffett Field CA, AIAA Member
[‡]Assistant Professor, Dept. of Mechanical and Aerospace Engineering, Utah State University, Logan UT, AIAA Member
[§]Assistant Professor, Dept. of Mechanical Engineering, University of Wyoming, Laramie WY, AIAA Member
[¶]Principal Research Engineer, Dept. of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge MA, AIAA Member

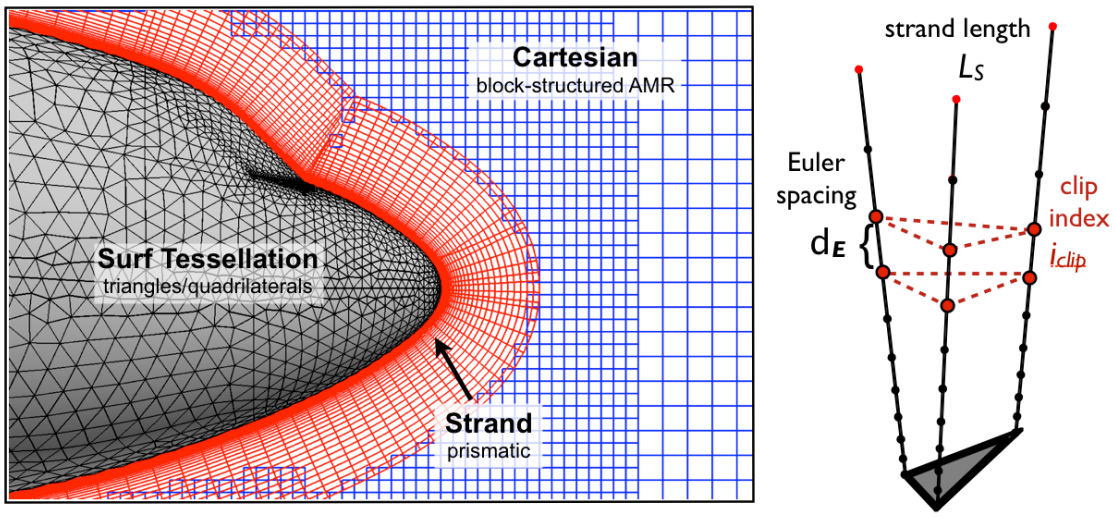


Figure 1. Strand-Cartesian grid system.

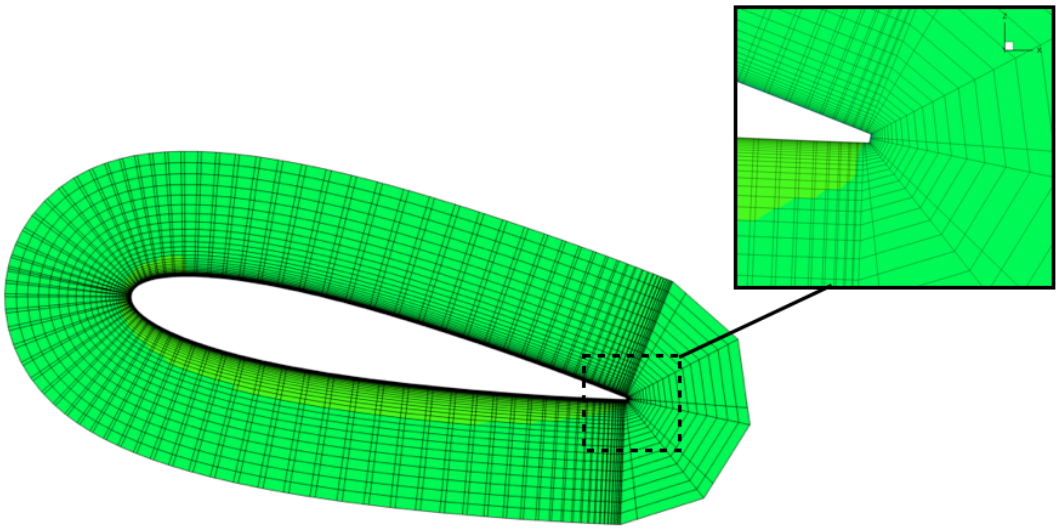
a parallel environment. This allows for significant gains in efficiency and scalability of domain connectivity, effectively eliminating inter-processor search routines. The savings become even more significant in the case of moving body simulations for which domain connectivity must be re-established at each unsteady time-step. Second, both strand and Cartesian meshes possess at least some grid structure, facilitating efficient implementations of high-order accurate discretizations and solution methods. These methods include high-order finite differencing, line-implicit solvers, and directional multigrid coarsening. Third, both the strand and Cartesian grids easily permit use of Adaptive Mesh Refinement (AMR). Because all strands use the same normal point distribution, adaptation is entirely surface-based. This avoids cell quality and edge swapping complexities that have traditionally plagued volume-based unstructured AMR. AMR on Cartesian grids has been known for years to be very effective because the logical data structure naturally facilitates a hierarchical mesh representation and Cartesian cells do not suffer cell quality issues with frequent and persistent adaptation, as can occur with tetrahedral elements.

Previous work introduced the mesh generation concepts and presented results for low-Re laminar flows computed on strand meshes. This paper extends the previous work by including RANS with the Spalart Allmaras turbulence model (RANS-SA) on strand meshes. Strand-Cartesian volume meshes are automatically constructed at runtime from an input surface mesh. The first part of the paper summarizes the automatic mesh generation paradigm in Section II, and presents the RANS-SA solver with validations on these strand meshes in Section III. Parallel implementation and scalability are presented in Section IV. With more complex 3D geometries the traditional strand grid generation approach used today can break down at highly convex geometric nodes and edges, such as a trailing edge, and at geometric saddle points, such as at the trailing region of a wing-body junction. The second part of the paper presents a new strand mesh generation procedure that addresses these deficiencies by using multiple strands emitted from a from a *solid* model, such as geometry from a modern CAD system, around geometric nodes and edges. The new multi-strand strand mesh generation procedure is outlined in Section V with sample 2D RANS-SA calculations performed on these automatically generated meshes shown in Section VI. Concluding remarks are presented in Sec. VII.

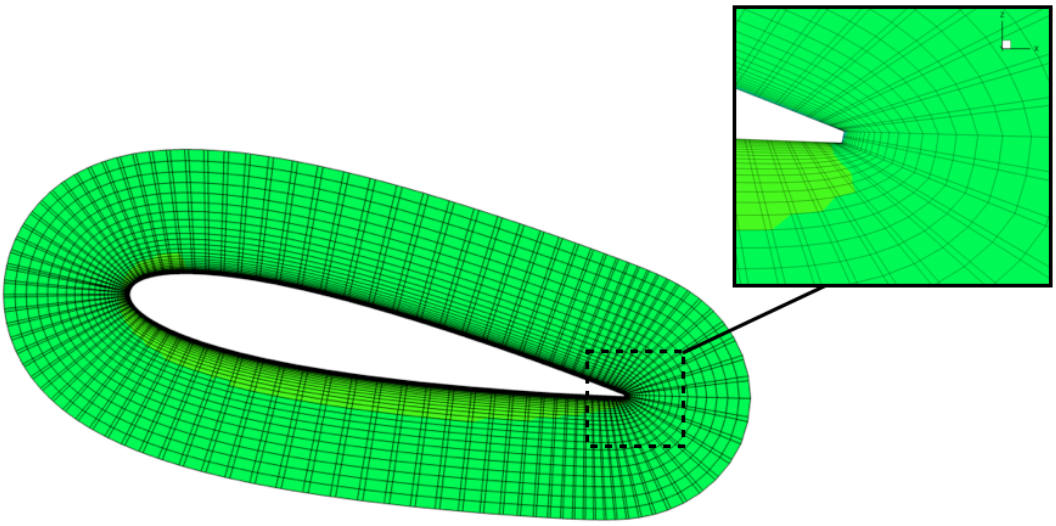
II. Strand Mesh Generation

The starting point for the mesh generation is a tessellated surface composed of either triangles, quadrilaterals, or a mix of both. Strands consist of straight line segments of equal length, number of points, and point distribution, grown from surface vertices. The strands initially are grown normal to the surface and then smoothed to provide coverage in convex corners (Fig. 2a & b) and to push crossing strands away from the surface in concave corners (Fig. 2c & d). The desired degree of smoothing in the mesh may be adjusted at runtime through an input parameter.

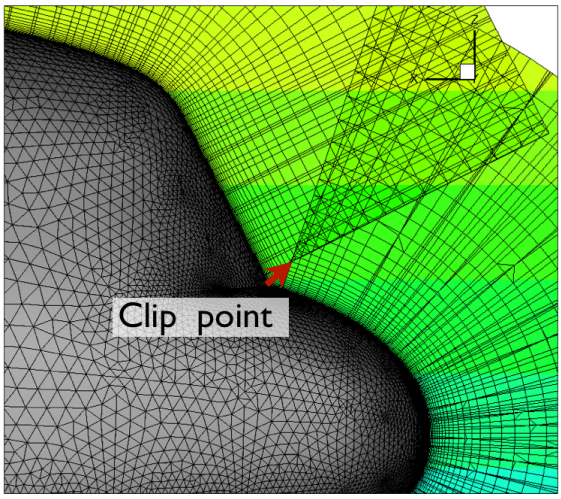
Spacing along each strand ranges from viscous at the root to transitional, or "Euler" spacing δ_E at the



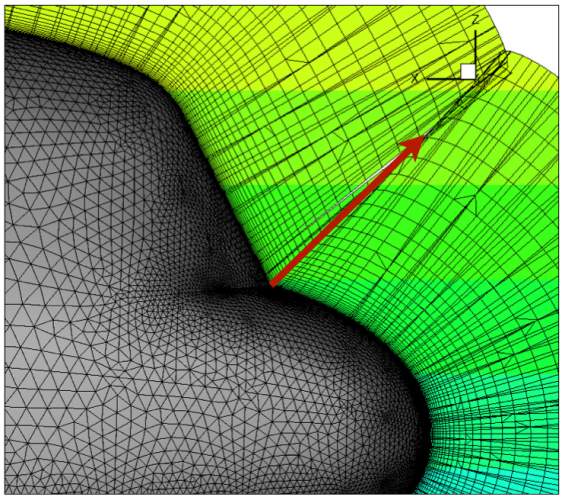
(a) Convex corner - Non-smoothed



(b) Convex corner - Smoothed



(c) Concave corner - Non-smoothed



(d) Concave corner - Smoothed

Figure 2. Strand direction vector smoothing.

tip. The set of strands produce a prism stack associated with each surface triangle. Any negative volumes associated with crossing strands are clipped by associating an integer "clip index" i_{clip} with each surface triangle. The clip index may also be used to clip elements that protrude an outer mold line for two strand meshes that lie in close proximity to one another. The user can supply the desired strand length L_S through input or allow the strand length to be computed automatically. If the length L_S is computed automatically, the algorithm seeks to make the transitional spacing δ_E at the strand end equal to the surface tessellation spacing. This ensures the transition cells are roughly isotropic at the strand ends so they transition nicely to Cartesian off-body meshes. The strand length L_S, Euler spacing δ_E, and clip index i_{clip}, are all pictured graphically in Figure 1.

Once a near-body strand mesh is available, an adaptive Cartesian off-body mesh is automatically constructed. The output of the near-body strand mesh generation is a set of prism stacks each with a designated clip index i_{clip} and the normal spacing of the prism element at this clip index, or Euler spacing δ_E, provides the basis for the initial Cartesian grid generation. The location of the clip index elements (x, y, z) and the Euler spacing δ_E are provided to the Cartesian grid generator. Block structured Cartesian grids are built in a hierarchical fashion, the coarsest level defines the physical extent of the computational domain and new levels are constructed from coarsest to finest. Each finer level is formed by selecting cells on the coarser level and then clustering the marked cells together to form block regions that will constitute the new finer level. The Cartesian grid is initially refined to match the Euler spacing δ_E elements in the strand mesh, and Cartesian grids are then adapted throughout the simulation to capture time-dependent solution features such as vorticity. Further details of the off-body mesh generation procedure have been presented in previous work.[13]

Visibility issues arise at geometric saddle points with the single strand per surface vertex paradigm. Later, in section V we present a proposed approach that applies multiple strands per surface vertex to address both the visibility issue as well as to provide better coverage in highly-convex regions such as at trailing edges. The next sections describe the 3D RANS solver applied to the current strand meshes.

III. RANS-SA Solver

In this work we solve the Reynolds-averaged Navier-Stokes (RANS) equations in three dimensions. Turbulence closure is accomplished with the Spalart-Allmaras (SA) model.[14] The RANS-SA equations may be expressed as

$$\frac{\partial Q}{\partial t} + \frac{\partial F_j}{\partial x_j} - \frac{\partial F_j^v}{\partial x_j} = S, \tag{1}$$

where the conserved variables, Q, inviscid fluxes, F_j, viscous fluxes, F_j^v, and source term, S, are defined as

$$Q = \begin{pmatrix} \rho \\ \rho u_i \\ \rho e \\ \rho\tilde{\nu} \end{pmatrix}, \quad F_j = \begin{pmatrix} \rho u_j \\ \rho u_i u_j + p\delta_{ij} \\ \rho h u_j \\ \rho\tilde{\nu} u_j \end{pmatrix}, \quad F_j^v = \begin{pmatrix} 0 \\ \sigma_{ij} \\ \sigma_{ij}u_i - q_j \\ \frac{\eta}{\sigma}\frac{\partial\tilde{\nu}}{\partial x_j} \end{pmatrix}, \quad S = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \mathcal{P} - \mathcal{D} + C_{b2}\rho\frac{\partial\tilde{\nu}}{\partial x_k}\frac{\partial\tilde{\nu}}{\partial x_k} \end{pmatrix}. \tag{2}$$

Here, ρ is the density, u_i is the Cartesian velocity vector, e is the total energy per unit mass, $\tilde{\nu}$ is the turbulence working variable, p is the pressure, h is the total enthalpy per unit mass, σ_{ij} is the deviatoric stress tensor, q_j is the heat flux vector, and η/σ is the turbulent diffusion coefficient. The turbulent source term consists of a production term, $\mathcal{P}$, and a destruction term $\mathcal{D}$. The stress tensor is defined as

$$\sigma_{ij} = 2(\mu + \mu_T)s_{ij}, \tag{3}$$

where μ is the dynamic viscosity, μ_T is the turbulent viscosity, and s_{ij} is the rate of strain tensor, defined as

$$s_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}. \tag{4}$$

The heat flux vector is obtained with Fourier's Law,

$$q_j = -C_p\left(\frac{\mu}{Pr} + \frac{\mu_T}{Pr_T}\right)\frac{\partial T}{\partial x_j}, \tag{5}$$

where C_p is the specific heat, Pr is the Prandtl number, Pr_T is the turbulent Prandtl number, and T is the temperature. The ideal gas equation of state, $p = \rho RT$ is used to close the equations.

III.A. Discretization and Solution Methods

The strand grid spatial discretization is based on a cell-centered approach where the primary unknowns are located at the centroid of the prisms formed by adjacent strands. The solver accommodates both quadrilateral and triangular prisms depending on the surface topology. However, control volumes are composed entirely of triangular facets by triangulating any non-planar quadrilateral faces. This is important for second-order accuracy on general prismatic grids with no assumption of underlying smoothness.[15] Linear reconstruction is employed to obtain second-order accuracy through first obtaining consistent nodal values of the conserved variables from surrounding cell-center values. A projection method is used to obtain these nodal values via least squares interpolation in a regression plane through the three-dimensional stencil of cells surrounding a strand.[15] Once the nodal values have been obtained, a Green-Gauss surface integration procedure is performed to obtain cell gradients in each control volume.

Inviscid fluxes rely on a reconstruction upwind formula for the numerical flux based on the approximate Riemann solver of Roe,[16]

$$\hat{\mathcal{F}} = \frac{1}{2}\left(\mathcal{F}(Q_R) + \mathcal{F}(Q_L)\right) - \frac{1}{2}\left|A(Q_R, Q_L)\right|\left(Q_R - Q_L\right), \tag{6}$$

where $\mathcal{F} = F_j n_j$ is the directed flux at a face with normal n_j, and $A = \partial\mathcal{F}/\partial Q$ is the directed flux Jacobian. The viscous terms are computed using values of Q and $\boldsymbol{\nabla} Q$ determined at each face,

$$\mathcal{F}^v = \mathcal{F}^v(Q_f, \boldsymbol{\nabla} Q_f), \tag{7}$$

where f refers to the face reconstructed values. These face values are easily obtained once nodal values have been reconstructed using the projection method described above. This method is similar to the node averaging schemes outlined by Diskin, et al.[17] Both the inviscid and viscous discretization methods described herein have been verified to be second-order accurate for arbitrary prismatic meshes under a variety of conditions using the method of manufactured solutions.[15]

The result of the spatial discretization of the viscous and inviscid fluxes is a coupled set of non-linear equations. In this work, we adopt a pseudo-time framework to march the steady or unsteady discretized equations to steady-state,

$$V\frac{\partial Q}{\partial \tau_k} + R(Q) = 0. \tag{8}$$

Here, V is the cell volume, and τ_k is the pseudo-time variable. The residual, $R(Q)$, contains the inviscid and viscous flux balances at each cell based on the cell-center discretization schemes described above. In order to reach a pseudo-steady state using an implicit scheme, the residual is linearized, leading to the following linear system to be solved at each pseudo-time step:

$$\left[\frac{V}{\Delta\tau_k}I + \frac{\partial R^k}{\partial Q}\right]\left(Q^{k+1} - Q^k\right) = -R(Q^k). \tag{9}$$

Here, $\partial R^k/\partial Q$ is the Jacobian of the residual. The linear system in Equation 8 in general is large and sparse, rendering direct inversion impractical. Iterative line Gauss-Seidel (GS) methods are employed to solve this system, where contributions along strands are collected to form a tridiagonal system. To facilitate the line GS iterations and to increase robustness, we introduce an additional "linear time" variable, τ_l,

$$V\frac{\partial Q}{\partial \tau_l} + \left[\frac{V}{\Delta\tau_k}I + \frac{\partial R^k}{\partial Q}\right]\left(Q^{k+1} - Q^k\right) = -R(Q^k). \tag{10}$$

The linear time is introduced to improve the diagonal dominance of the line GS procedure in order to increase robustness. Rearranging Equation 10 in terms of solution updates in linear time results in

$$\left[\left(\frac{1}{\Delta\tau_k} + \frac{1}{\Delta\tau_l}\right)VI + \frac{\partial R^k}{\partial Q}\right]\left(Q^{l+1} - Q^l\right) = -R(Q^k) - \left[\frac{V}{\Delta\tau_k}I + \frac{\partial R^k}{\partial Q}\right]\left(Q^l - Q^k\right). \tag{11}$$

Upon convergence of the linear iterations in l, the linear system of Equation 9 is satisfied. At that point, the next pseudo-time step in k proceeds. When the pseudo-time iterations converge, then the residual equation $R(Q)$ is satisfied for a given physical time station. All Jacobian terms in this work are first order and retain only nearest neighbor contributions. Further details of the implicit solution method may be found in previous work.[11]

III.B. Turbulence Model

The standard SA model is used when the turbulent working variable is positive. Details of the positive model, including the well-known definitions of the production and destruction terms, may be found in the original work by Spalart and Allmaras.[14] Modifications to the model to accommodate negative values of the turbulence working variable have been suggested recently by Allmaras[18] and are employed in this work. In the case of negative values of $\tilde{\nu}$, the following turbulence equation replaces the standard model:

$$\frac{\partial \tilde{\nu}}{\partial t} + u_j \frac{\partial \tilde{\nu}}{\partial x_j} = C_{b1}(1 - C_{t3})\Omega\tilde{\nu} + C_{w1}\left(\frac{\tilde{\nu}}{d}\right)^2 + \frac{1}{\sigma}\left[\frac{\partial}{\partial x_j}\left((\nu + \tilde{\nu}f_n)\frac{\partial \tilde{\nu}}{\partial x_j}\right) + C_{b2}\frac{\partial \tilde{\nu}}{\partial x_k}\frac{\partial \tilde{\nu}}{\partial x_k}\right], \qquad (12)$$

where,

$$f_n = \frac{C_{n1} + \chi^3}{C_{n1} - \chi^3}, \quad (C_{n1} = 16).$$

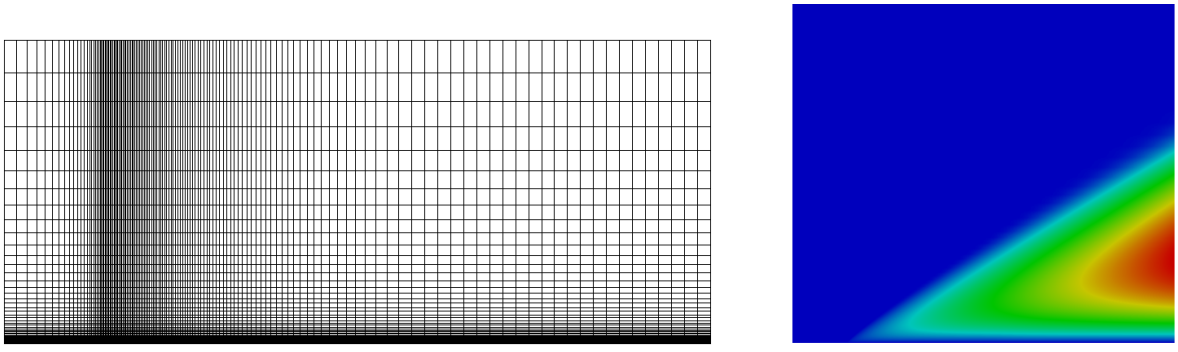
Here, Ω is the vorticity magnitude, d is the distance to the nearest wall, and $\chi = \tilde{\nu}/\nu$ is the ratio of the turbulent working variable to the kinematic viscosity of the fluid. All other constants in Equation 12 take the values found in the standard model.

III.C. Turbulence Model Validation

The S-A implementation in the strand solver is validated for flow over a flat plate at $M = 0.2$ and $Re = 5{\times}10^6$, based on a plate of length unity. This case was taken from the NASA Langley turbulence modeling resource has been made for this case.[19]

The grid used for the flat plate validation is a 137×97 grid shown in Figure 3(a). The plate leading edge begins at $x = 0$ and extends for a length of two. A short inviscid wall entry way beginning at $x = -0.33$ is provided to allow for proper inflow conditions. Stagnation temperature and pressure are specified at the inflow, and static pressure is specified at the outflow. The turbulent viscosity field for this case is shown in Figure 3(b), which has been scaled by a factor of 40 vertically to facilitate visualization. Streamwise velocity and turbulent viscosity profiles are shown in Figure 4(a) and 4(b) for two locations downstream on the plate, and are over plotted with FUN3D and CFL3D results. Note that good agreement is obtained, even for this 137×97 grid which is 16 times more coarse than the FUN3D and CFL3D results in the figures. The computed drag coefficient, which is entirely due to skin friction for this case, is shown in Table 1, along with FUN3D and CFL3D results for the same grid. The drag coefficient falls within the range predicted by the established codes.

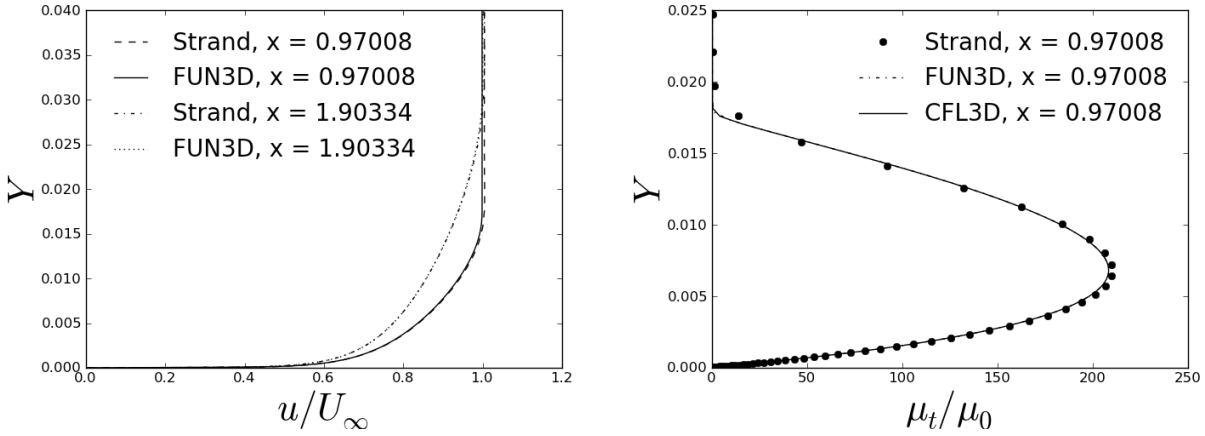
The next validation case is 2D flow over a NACA 0012 airfoil at $M = 0.15$ and $Re = 6 \times 10^6$ at various angles of attack. The strand grid shown in Figure 5 consists of a smoothed NACA 0012 airfoil containing 320 surface nodes and 64 cells along each strand, yielding 20,480 total strand cells. Figures 6 and 7 show lift coefficient versus angle of attack and drag coefficient versus lift coefficient, respectively, along with the corresponding experimental data of Ladson.[20] The lift data is matched reasonably well for all angles of attack, although a slight over-prediction of lift is observed for the high-α $\alpha = 15^o$ case. Drag results are



(a) 137x97 grid

(b) Contours of μ_t/μ_0

Figure 3. Grid and turbulent viscosity contours for flow over a flat plate at $M = 0.2$ and $Re = 5 \times 10^6$.



(a) x-velocity profiles (b) turbulent viscosity profile

Figure 4. Comparison of streamwise velocity and turbulent viscosity profiles for flow over a flat plate at $M = 0.2$ and $Re = 5 \times 10^6$.

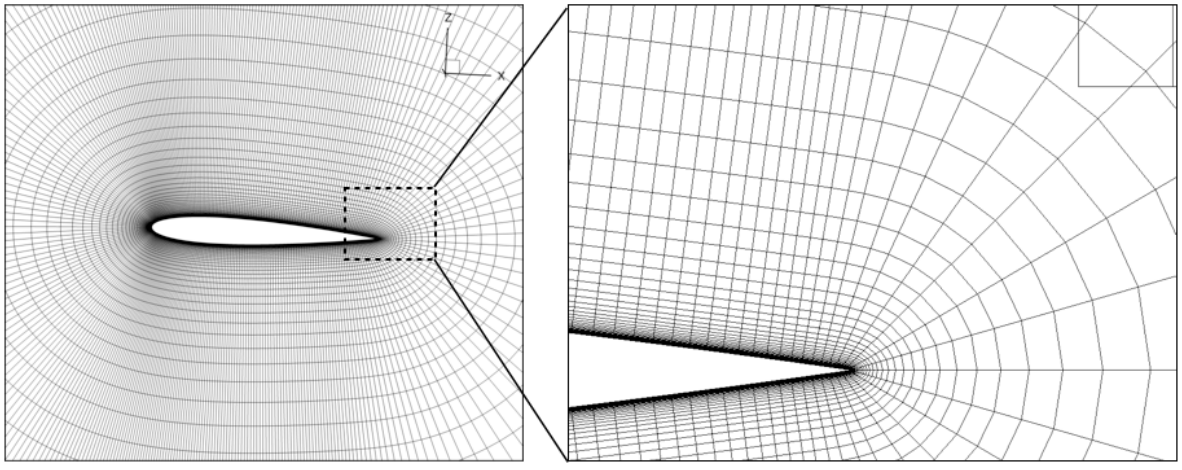


Figure 5. 137×97 **NACA 0012 strand grid**

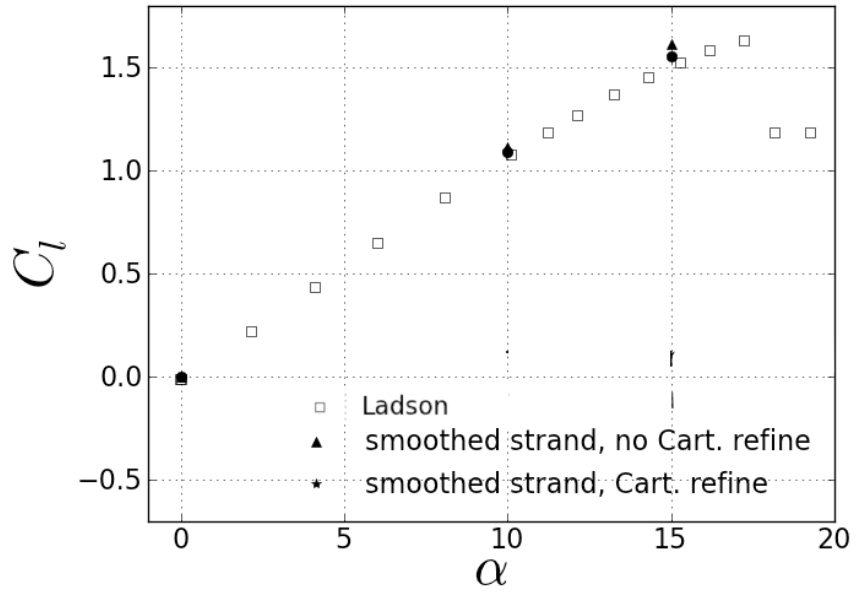


Figure 6. C_l **vs.** α **compared to experiment for flow over NACA 0012 airfoil at** $M = 0.15$ **and** $Re = 6 \times 10^6$**.**

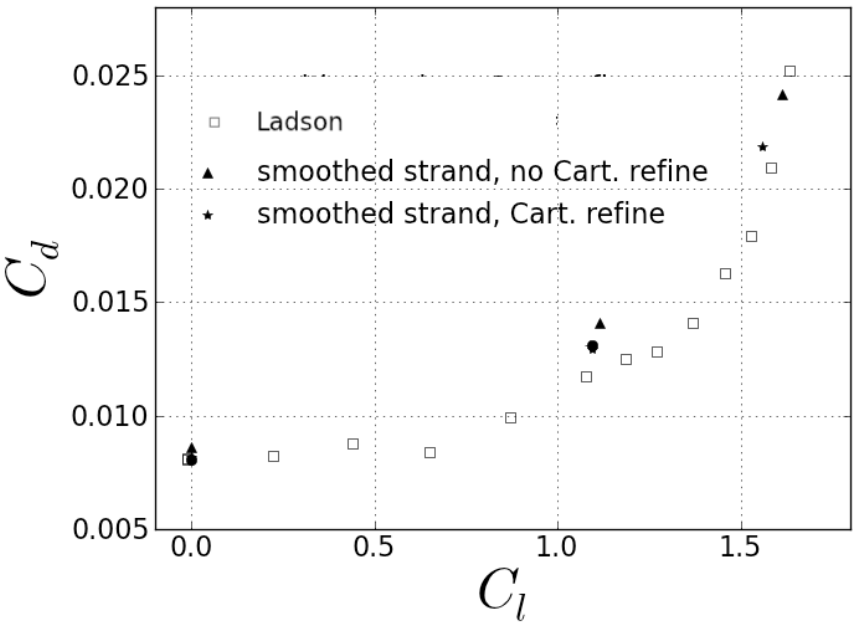


Figure 7. C_l **vs.** C_d **compared to experiment for flow over NACA 0012 airfoil at** $M = 0.15$ **and** $Re = 6 \times 10^6$**.**

shown in Figure 7 with the $\alpha = 15^o$ results summarized in Table 2. FUN3D and CFL3D results using a much finer grid (513 surface nodes instead of 320) are included for comparison. In general the strand solver results fall well within the range of these well established codes.

IV. Parallel Implementation

The strand mesh is partitioned for parallel execution by applying the ParMetis[21] software to the surface tessellation. Strands are sprouted from the partitioned surface separately on each strand block. See Fig. 8.

An advantage of surface-only partitioning is that it is simpler and less computationally time consuming than 3D volume partitioning so it is easier to repartition when surface-based adaptivity is applied. This approach also enables the use of implicit Gauss-Seidel-like schemes in the normal strand direction. Lastly, the surface-based partitioning better ensures scalability in overset domain connectivity between the strand and Cartesian meshes. Domain connectivity computations tend to scale with the number of interface points between overset meshes. Hence, the best parallel partitioning strategy attempts to evenly distribute the interface points. In the strand paradigm, the interface or clipped strand points align directly with the surface points so surface-based partitioning implicitly produces the best partitioning for domain connectivity. Tests reveal that domain connectivity scales commensurately with the flow solver.[22]

The disadvantage of surface-based partitioning is that it has the potential to make the communication to computation ratio non-ideal, since the volume to surface area ratio of each block is not minimized. This results in potentially additional communication in the flow solver. However, as shown in Section IV, scalability tests on the code that uses this partitioning strategy reveal good scaling so far. On very large number of processors the "sliver" partitions potentially created by surface-only partitioning may introduce performance issues but, at present, the advantages of surface-only partitioning appear to outweigh any loss in parallel performance from non-ideal partitions.

The PICASSO infrastructure[13] supports the runtime strand and Cartesian mesh generation, adaptation, partitioning, parallel communication, and load balancing for solvers that operate on this meshing paradigm. The library constructs the strand and Cartesian mesh blocks, allocates space for data to be stored on the blocks, and provides communication operations to exchange data between blocks on a distributed memory parallel computer system. It is written in C++ with published interfaces to support single-block numerical solvers that may be plugged in to the library. A solver developer may write underlying numerical kernels in any language preferred, whether its Fortran, C, or C++.

Parallel scaling qualities of the RANS-SA solver described above implemented within PICASSO are evaluated on the Maui High Performance Computing Center (MHPCC) *mana* system which has computational nodes consisting of a Dell PowerEdge M610 series, each with two 2.8 GHz quad-core Intel Nehalem processors and 24 GB RAM (i.e. each node contains 8 cores with 3GB/core memory), and Dual Data Rate Infiniband interconnect fabric. The problem used for the scaling study is 3D symmetric viscous NACA 0012 airfoil case at angle of attack $\alpha = 3^o$, $M = 0.5$, and $Re = 5000$.

Computational performance is assessed on three quad-only meshes, coarse, medium, and fine. The three meshes differ only in their surface resolution, the same strand normal distribution of 64 nodes is used for all three. Grid statistics are given in Table 3. Note that the medium mesh has exactly 4X the number of gridcells as the coarse mesh, and the fine mesh has exactly 16X the number of gridcells as the coarse, and 4X the medium, respectively. A colored Gauss-Seidel algorithm is used to converge the implicit solution. Plots of the meshes and Mach contours of the final converged solution for the coarse, medium, and fine meshes are shown in Figure 9.

Measured computational performance for the cases run on up to 64 cores is shown in Table 4. Both strong and weak scaling characteristics are evaluated. The coarse grid is run on 1, 2, and 4 cores; the medium case

	C_d
Strand	2.82287E-3
FUN3D (quads)	2.84005E-3
FUN3D (triangles)	2.80289E-3
CFL3D	2.86621E-3

Table 1. Comparison of computed drag coefficients for flow over a flat plate at $M = 0.2$ and $Re = 5 \times 10^6$.

on 4, 8, and 16 cores; and the fine case on 16, 32, and 64 cores. Strong scaling - comparing how the solution time varies with the number of processors for *fixed problem size* - is assessed for each case by doubling the number of cores the case is run on, twice. Weak scaling - comparing how the solution time varies with the number of processors for *fixed problem size per processor* - is assessed by comparing the performance at each doubling, since the medium and fine cases are 4X and 16X times as many grid cells as the coarse case, respectively. A plot of the scaling factor is shown in Figure 10.

The weak scaling qualities of the algorithm are quite good. Ideal scaling would result if the time per step remained the same for the coarse grid run on 1 core, the medium grid on 4 cores, and the fine on 16 cores. In fact we see the cost actually decrease indicating superlinear scaling. The strong scaling qualities are also quite good, particularly on 2, 4, and 8 cores. Cases which run on more than 8 cores invoke inter-node communication, which is likely why there is a relative dropoff in efficiency on 16, 32, and 64 cores. No effort was made to overlap communication with computation for this study, the results reveal this would be an useful idea to consider in future work.

Convergence of the colored Gauss-Seidel algorithm is essentially independent of the number of processors used. Figure 11 shows the convergence of $\Delta\rho/\rho$ for the three mesh systems. Although the convergence rate is dependent on the problem size – the coarse case converges in 1000 steps, medium in 4000 steps, and fine in 16000 steps – it is independent of the number of processors.

V. Multi-Strand Mesh Generation

The goal of strand/Cartesian mesh generation is to automate viscous mesh construction directly from the geometry source (for example, a CAD system). This allows for an analysis process that can be used in design settings if the grid generation can be performed with minimal human intervention. To facilitate full automation the best starting point is a *solid* Boundary Representation as can be constructed in current Computer-aided design (CAD) systems. This type of data is transmitted using the Standard for the Exchange of Product model data (STEP) format, as can be seen in Fig. 12(a). The STEP file provides the BRep entities which are divided into topology and geometry. The topological entities include Faces, Edges and Nodes each of which contains the geometry entities of surfaces, curves and points, respectively. The topology is used to *close* the model and is necessary because the trimming curves that bound Faces may not lie on either Face just as the Nodes that bound the Edges may not be on the underlying curve.

V.A. Near-body Meshing Procedure

1. Tessellation consistent with the Geometry Kernel/CAD System
   Since the strands emanate from a tessellation of the body of interest, it is important that whatever scheme is used to provide that discrete view of the geometry either comes from, or can be reassociated with, the BRep as held by the geometry kernel. The tessellation should be watertight and manifold, contain only triangles and/or quadrilaterals, Edges are traceable and must begin and end at vertices that are the location for Nodes, and all tessellation vertices are marked with the owning geometry (Faces, Edges and Nodes). Also, a list of Faces and $[u, v]$ coordinates for each Face is required at any vertex in the tessellation. The list has a single entity for a vertex interior to a Face, usually 2 for an Edge and there are 2 or more entries in the list for Nodes.

   Fig. 12(b) displays the geometry that is automatically tessellated into triangles and seen in Fig. 12(c). The desired edge spacing δ_s for the triangulation, as well as parameters that control curvature-based resolution and resolution of small geometric features, effect the resulting tessellation.

   The list of Faces can be used to compute the surface normals that are required for each vertex. These

	C_l	C_d
Smoothed strand (320 points)	1.5578	0.02189
FUN3D (513 points)	1.5547	0.02159
CFL3D (513 points)	1.5461	0.02124

Table 2. Comparison of computed lift and drag coefficients for flow over a NACA 0012 airfoil at $M = 0.15$, $\alpha = 15^o$ $Re = 6 \times 10^6$. Note that the FUN3D and CFL3D results use a much finer grid.

Table 3. Strand grids used for parallel scaling study

Grid	surf nodes	volume grid nodes	volume grid cells
coarse	160 × 41	0.419M	0.403M
medium	320 × 81	1.659M	1.613M
fine	640 × 161	6.595M	6.451M

Table 4. Parallel Timings for three different mesh sizes. Time/step shown in seconds.

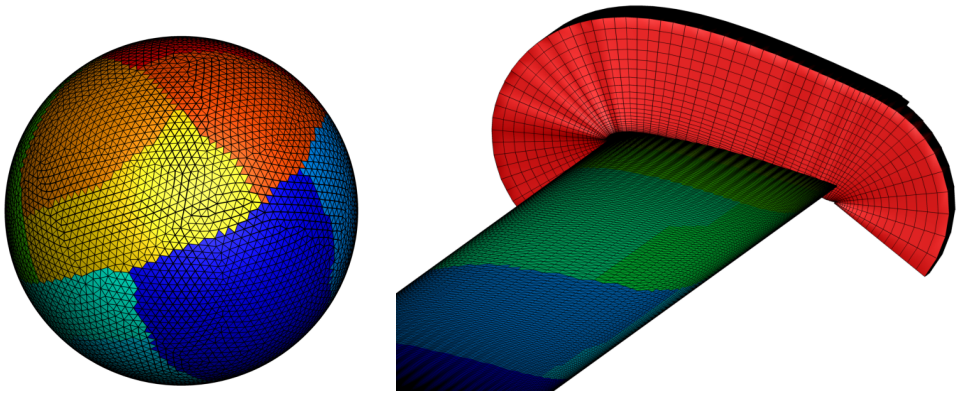
	coarse		**medium**		**fine**	
cores	time/step	par eff	time/step	par eff	time/step	par eff
1	9.716	100%				
2	4.920	99%				
4	2.379	102%	9.266	100%		
8			4.840	96%		
16			2.550	91%	9.698	100%
32					5.126	95%
64					3.028	80%

will represent the starting positions for the strands. Those vertices associated with Edges and Nodes will initially have multiple normal specifications (one for each Face touching the entity).

2. Classification of vertices associated with Edges/Nodes
The classification of vertices is used to determine how to treat specific situations found in the geometry. It drives whether the normals can be merged into a single strand or if adding additional strands at an Edge/Node vertex is required.

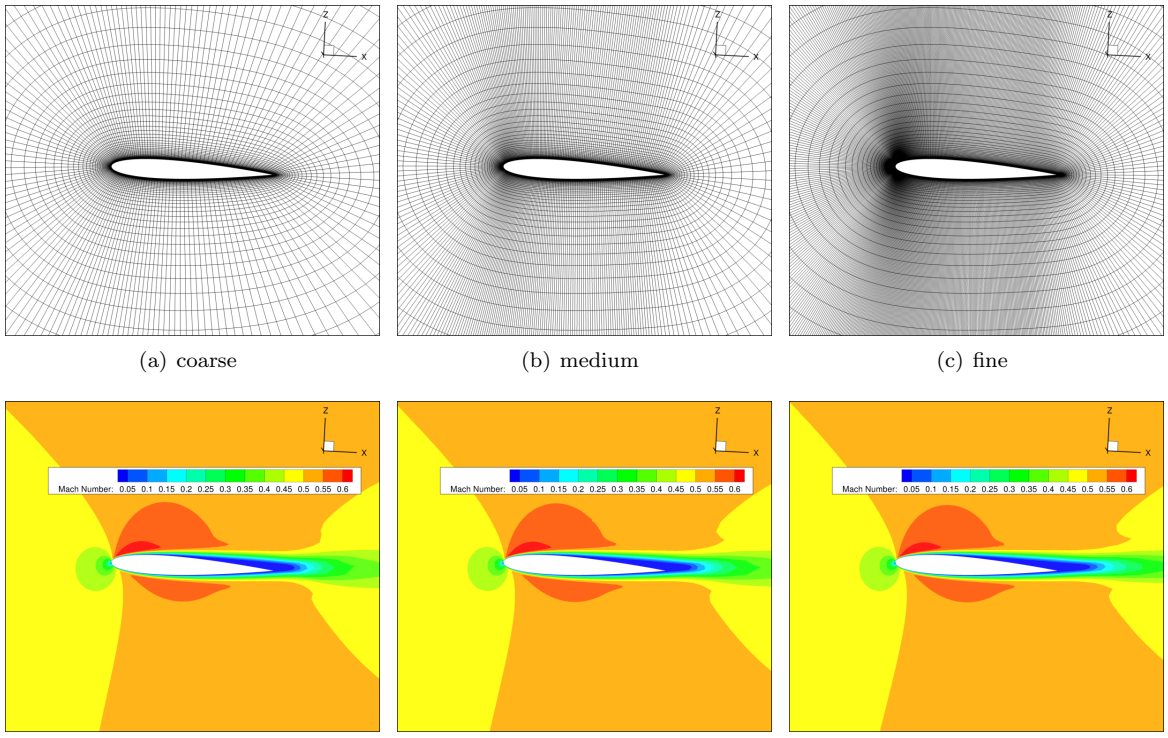
   The classification is driven by the *winding angle* found between pairs of Faces and the normals associated with each Face. The *winding angle* is simply the angle traversed starting at one surface, pivoting at the vertex, and ending at the other (in the plane generated by the cross product of the 2 normals). If the angle is less than 180^o then the pair is *concave*, if greater then it is considered *convex*.

   - *Convex Edge Vertex*. This is where the pair of Faces is convex, which may require adding additional strands by fanning unless the vertex can be also classified as *Same Normal*.



(a) surface-based partitioning (b) strands grown on each partition

Figure 8. Surface-based partitioning for strand meshes.



(a) coarse (b) medium (c) fine

(d) coarse (e) medium (f) fine

Figure 9. NACA 0012 3D symmetric wing grid and solution, used for parallel scaling studies; *top* meshes, *bottom* computed Mach contours.

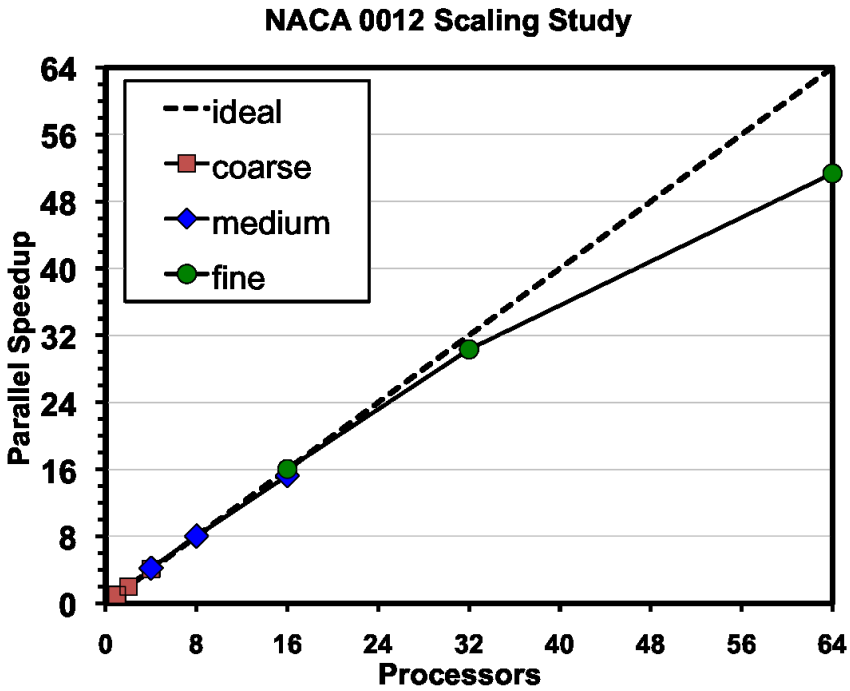
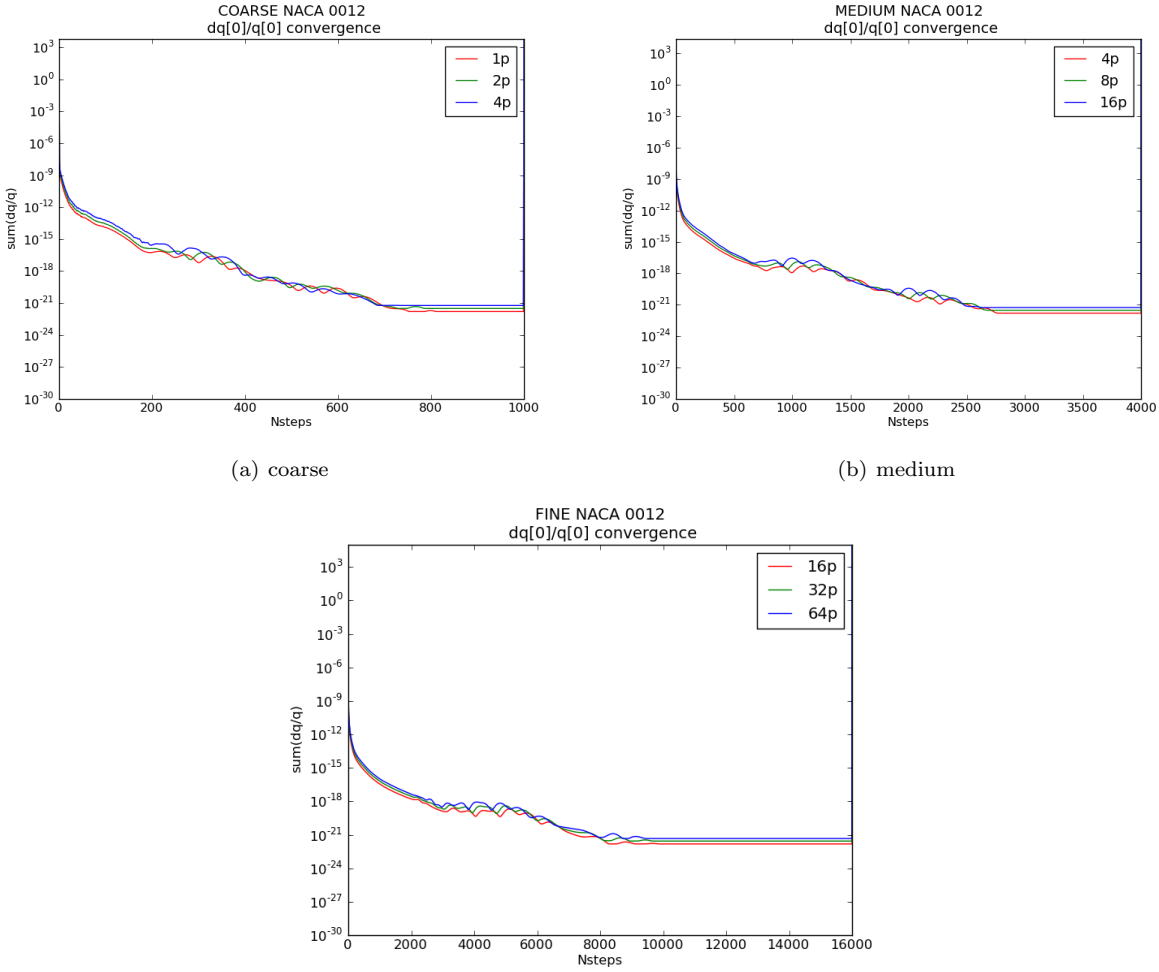


Figure 10. Measured parallel speedup using three different mesh sizes.

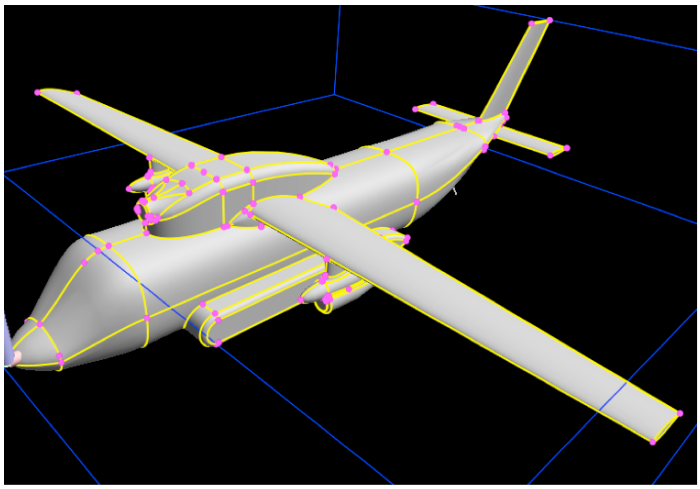


(a) coarse

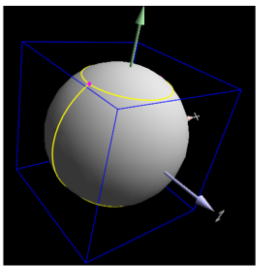
(b) medium

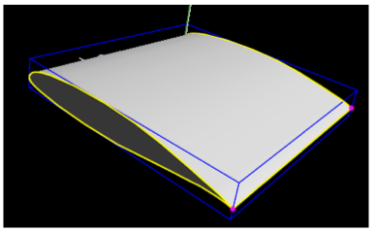
(c) fine

Figure 11. 3D NACA 0012 convergence of density residual $\Delta\rho/\rho$ using coarse, medium, and fine meshes on different number of processors.

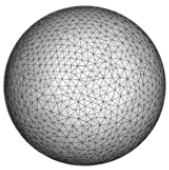


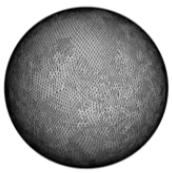
(a) STEP file import from CAD

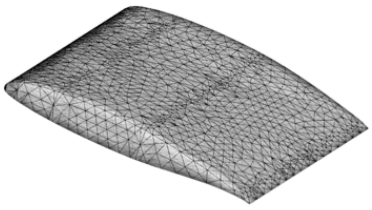




(b) Boundary Representation







(c) Surface Tessellation

Figure 12. Solid model geometry. (a) Import of a full vehicle via STEP; (b) BRep where yellow lines are the Edges and red points are the Nodes; (c) automatically generated surface tessellations at different resolutions.

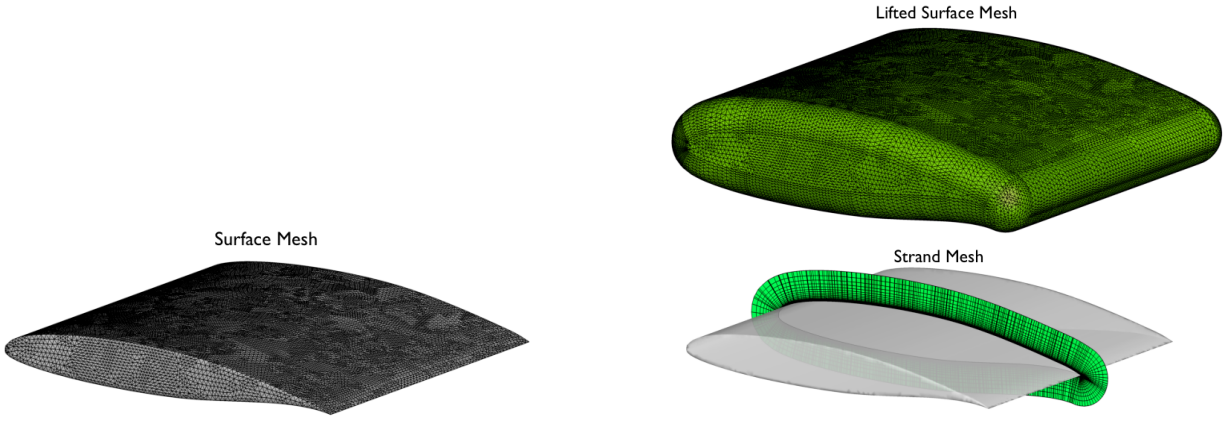
- *Concave Edge Vertex*. If the Edge vertex displays a Face pair as concave then it is marked for merging.
- *Convex Node*. This vertex has all combinations of pairs of Faces (from the Face list) indicating convex.
- *Concave Node*. This vertex has at least a single Face pair flagged as concave.
- Same Normal. When the difference between the normals for a Face pair is less than 3^o, the vertex is marked as having the same normals. This occurs when there is a smooth transition between Faces or the vertex is from a periodic-like seam separating a single closed surface.

3. Merging of normals for a *Concave Edge/Node* strand
   Marking a vertex as being either a *Concave Edge* or a *Concave Node* indicates that the multiple normals will be coalesced to a single strand. The direction of the strand is the average of the directions of all of the normals in the Face list. Any set of normals that is marked as *Same Normal* will only be included once in the sum. The new strand direction is renormalized.

4. Specifying Edge strand fanning numbers
   Each vertex marked as a *Convex Edge* has the tessellation on the Face pair examined for the local spacing perpendicular to the Edge itself. The spacing on both Faces is averaged and by knowing the *winding angle* and the strand length it is simple to compute the number of subdivisions of the *winding angle* that are needed to meet that average spacing requirement. This number is stored away with the vertex.

   After all of the Edge vertices are handled, each complete discretized Edge is examined by traversing from start to end Node. Any abrupt changes in subdivision numbering is smoothed. Note that this will taper the subdivision numbers of any *Convex Edge* vertex adjacent to a *Concave Node* which obviously needs to collapse to the single strand.

5. Creation of Edge fanned strands
   For each vertex marked as a *Convex Edge*, new strands are created that fan from one of the Face normals to the other. These are the locations that will be connected by sets of triangles constructed for each discretized Edge segment.



(a) Surface tessellation (b) Lifted surface tessellation

Figure 13. Strand mesh generation producing a "lifted" surface.

6. Filling in of *Convex Nodes*
   The number of sides of this convex open polygonal region is determined by the number of Edges meeting at the Node. A new spacing requirement is set for each void as the average of the exposed segment distances. The void is closed by the following procedure:

   (a) Creation of center strand. This is done by averaging the direction of all of the strands that outline the convex region to be filled.

(b) Close up the exposed Edge segments by creating triangles that have 2 positions on the exposed Edge opening and connect to the new center strand.

(c) Insert a new strand where the spacing is too large by splitting the interior triangle side which creates 4 triangles from the original 2.

(d) Use a MINMAX angle criteria to drive swapping of interior triangles in order to maintain a good tessellation.

(e) Iterate on (c) and (d) above until the spacing requirement is satisfied.

The resultant *lifted surface* after filling the *Convex Node* vertices can be seen in Fig. 13(b). All element stacks from the original tessellation are a simple reflection of the surface and could produce series of prisms for triangles and stacks of hexahedra for quadrilaterals. All newly constructed triangles (at the *lifted surface*) do not exist in the original tessellation and collapse to a single surface vertex (associated with either a BRep Edge or a Node). The stack is primarily prismatic except at the base (the surface vertex) where it degenerates to a tetrahedron. A closer examination of the constructed regions can be seen in Fig. 14.

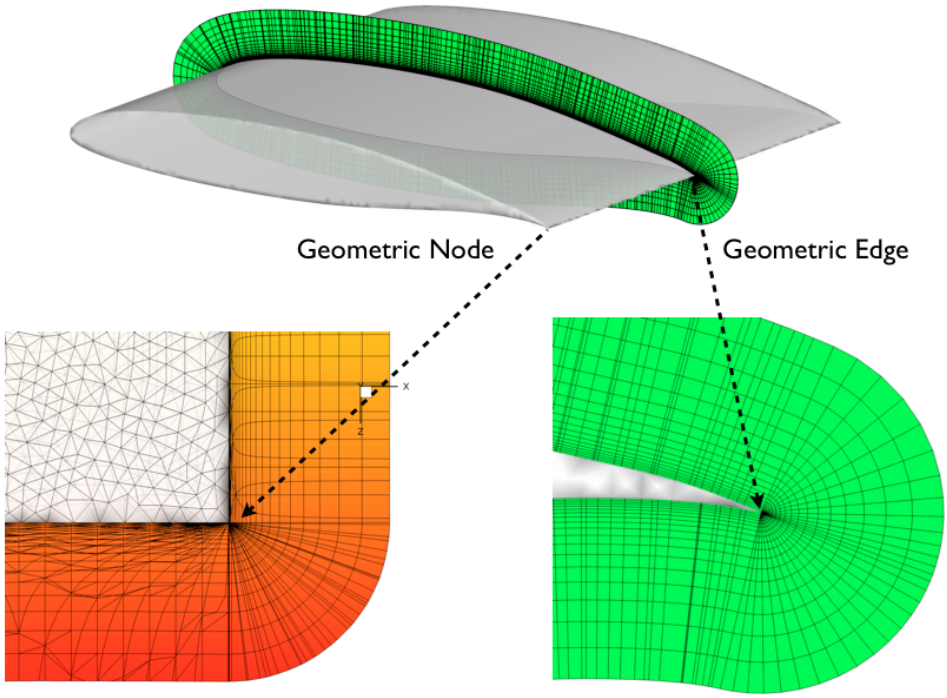


Figure 14. Multi-strands at BRep Nodes and Edges.

Concave corners or concave regions of high-curvature, such as at wing-body or nacelle junctions, often experience strand crossings, i.e. Fig. 2(c). Localized smoothing, or root-pivoting of the direction vectors, is applied, as in Fig. 2(d), to push the crossing points away from the surface. But unlike the original implementations, the Laplacian smoothing applied is selectively local. The procedure used is:

- Mark all strands where the stack validity check terminates early. Initially there should be no candidates from *Concave Edges/Nodes* due to their cylindrical and radial-like construction.
- Flood the *lifted surface* neighbors up to a specified depth away from marked strands. This allows for pivoting into a larger region.
- Update the touched strands by performing the Laplacian smoother (averaging neighboring strand directions and renormalizing).

Generally the strand smoothing is done in 2 phases, each is terminated by the convergence of strand directions or by reaching a maximum number of iterations. Each phase is performed a user specified number of times:

1. Edge/Node phase. This only adjusts strands emanating from either Edge or Node vertices.

2. Interior phase. Only smoothes strands that can be found interior to Faces.

This selective smoothing maintains strand orthogonality in regions where there are no collisions. An orthogonal mesh improves the accuracy of algebraic turbulence models, but more important is the effect on resolution. The strand length should be specified to be the maximum size expected of the boundary layer found in the simulation (this is a global quantity). This ensures that the boundary layer is properly covered and therefore properly formed. If the strands are significantly pivoted, then much longer strands are required to cover the same distance off the wall. This is a problem in regions where the boundary layer is small or just forming.

The last step in the process is to exclude any cells with: 1) negative volume prism cells, and 2) elements that protrude into the surface geometry outer mold line. This is done by the domain connectivity package[22] setting the clip index. The intent of the strand grid is to provide a transition region from the very high aspect ratio viscous boundary layer cells near the surface to isotropic Cartesian cells in the field.

V.B. Near-body Adaptation

The near-body strand mesh is not adapted in the current implementation, but adding adaptivity is fairly straightforward. By specifying triangle/quadrilateral element at the *lifted surface* as well as the barycentric coordinates in that element, the strand mesh can be adapted. If the triangle is constructed, then there is no explicit surface involvement, a new strand emanates from the single surface vertex so that the barycentric coordinates in the element at the *lifted surface* is pierced. If internal to the element, then 3 triangles are produced from the one. If the position is along a triangle side, then the neighboring element is also involved. If the neighbor is a triangle, then the 2 triangles are broken up along the shared side and 4 triangles result. If the neighbor is a quadrilateral, the triangle is broken into two and the quadrilateral is slit into 3 triangles.

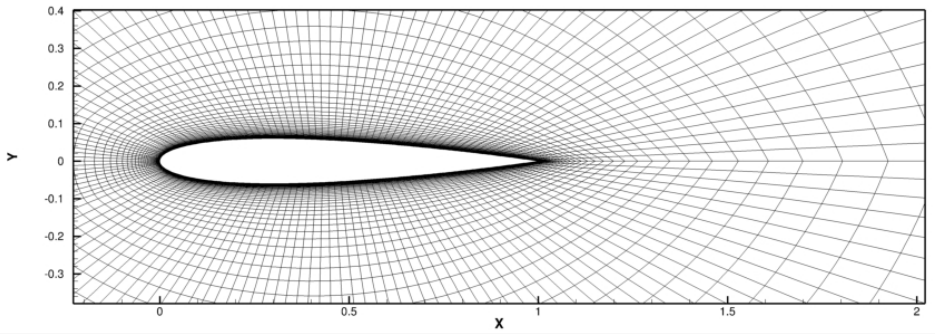
For element stacks that map to surface elements, the barycentric coordinates described at the *lifted surface* are used on the corresponding surface element to query the geometry kernel for the actual normal (at that point). The new strand is created and if the barycentric coordinates indicate an internal position, the stack is split into 3 for triangles or into 4 triangles (for a quadrilateral). If the position is on a side, then like above, the neighboring element is also involved where quadrilaterals are broken into 3 triangles. At this point the new stacks need to be checked for collisions and smoothing applied when needed.

VI. Multi-strand Solver Tests

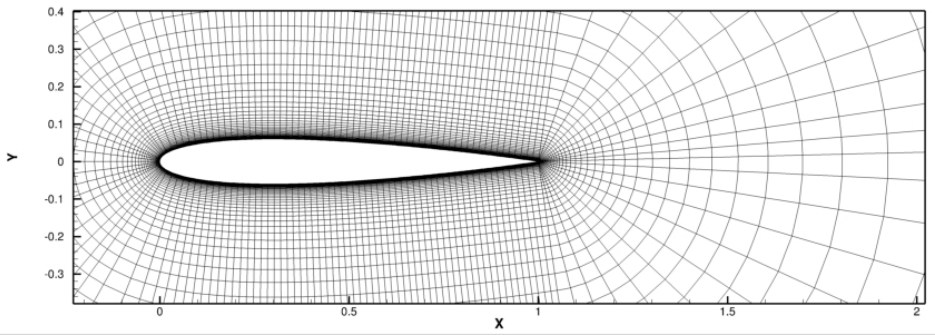
The multi-strand mesh generation procedure discussed above is designed to enable the robust and automatic creation of viscous volume meshes, near convex geometric features. The meshes generated by this approach are distinct from those generated using root bending due to the presence of non-prismatic elements in three dimensions (non-quadrilateral elements in two dimensions). Employing non-prismatic elements increases the complexity of the CFD solver employed for strand meshes, as the solver must now solve the equations on multiple element types. Multi-Element strand meshes also have the equivalent of a pole singularity at the trailing edge as multiple strands are emanating from a single grid node. Furthermore, many assumptions about the structure of stand meshes are violated by multi-strand meshes, such as the direction of grid anisotropy. Previous work[12] has documented the flow solutions obtained using root-bent strand meshes. Multi-strand meshes have received less attention in the literature. Therefore, one is motivated to compare the solutions obtained using root bent (simply referred to as bent-) and multi-strand meshes, in order to determine if the features/complexities of multi-strand meshes adversely impact solution accuracy.

A grid convergence study of the computed drag coefficient for the flow over a NACA0012 airfoil in two dimensions is conducted to quantitatively compare the solutions generated using bent- and multi-strand meshes. The flow conditions are, free stream Mach number $M_\infty = .15$, angle of attack $\alpha = 0.0^o$ and Reynolds number $Re = 6,000,000$. The equations of motion are the Reynolds Average Navier-Stokes (RANS) equations coupled to the negative Spalart-Allmaras version 2 turbulence model.[18] The goal of this study is to determine if the grid topology of multi-strand meshes induces an unacceptable amount of

discretization error in the drag coefficient. The equations are solved on a sequence of four near uniformly refined strand meshes, generated using both both- and multi-strand techniques. The meshes range in size from $N = 1,566$ to $N = 19,397$ elements. Figures 15(a) and 15(b) depict two of the meshes used in this study. The mesh in Figure 15(a) was generated with a bent-strand trailing edge treatment and the mesh in Figure 15(b) was generated using a multi-strand trailing edge treatment. For this test case the equations are solved using a discontinuous finite-element method (FEM) flow solver,[23–25] with a $p = 1$ or second-order accurate discretization.



(a) Example of bent-strand mesh, $N = 16,898$ elements



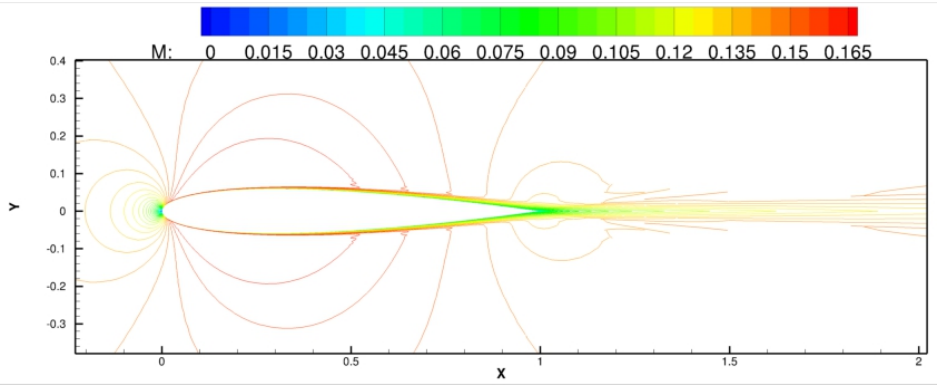
(b) Example of multi-strand mesh, $N = 19,397$ elements

Figure 15. Example bent and multi-strand meshes.

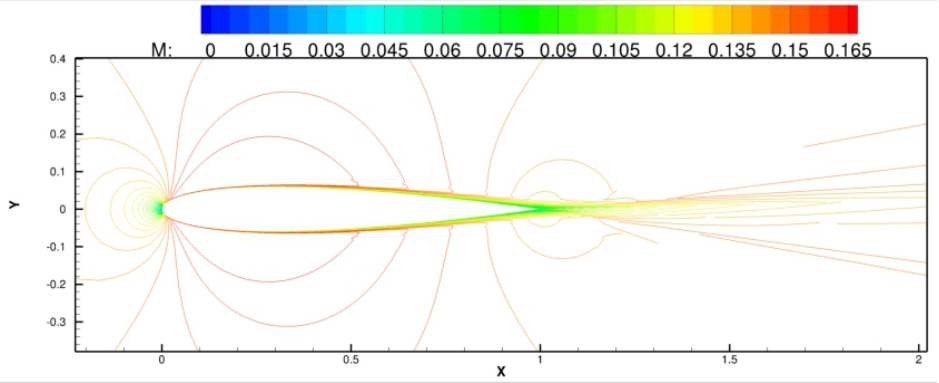
Figures 16(a) through 16(d) show the computed Mach number and turbulent eddy viscosity contours using the finest bent- and multi-strand meshes. These Figures give assurance that qualitatively both mesh generation techniques yield meshes on which the flow solver obtains reasonable results. Examination of these figures reveal that the only qualitative difference between the solutions obtained using bent- or mulit-strand meshes, is an asymmetry in the wake of the airfoil on the multi-strand mesh. The asymmetry of the wake is not surprising as the multi-strand mesh is not symmetric about the center of the wake.

Figure 17 which, shows the non-linear residual convergence vs. Newton iteration, clearly demonstrates that the non-linear residuals are converged at least 12 orders of magnitude. Therefore, algebraic error will not corrupt the grid convergence study.

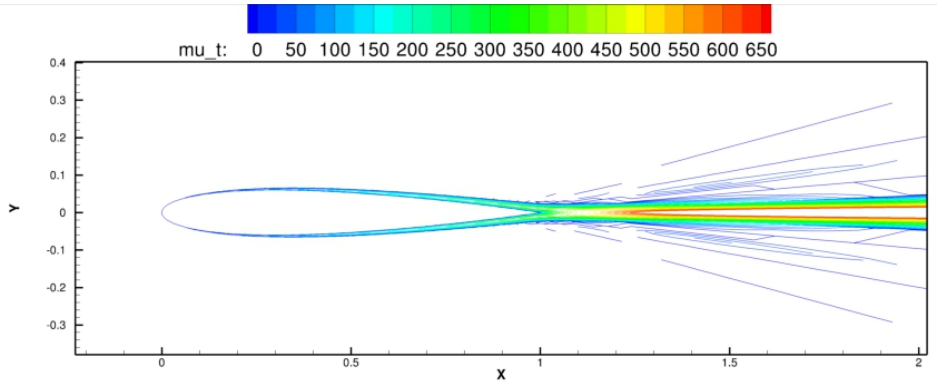
Furthermore, both types of meshes converge in similar numbers of non-linear iterations indicating that employing either mesh generation method does not induce undo stiffness for the linear/non-linear solvers. All coarser meshes display similar convergence behavior and are converged to as tight a non-linear residual tolerance. Figure 18 depicts the grid convergence of the computed drag coefficient for uniform refinement of both bent- and multi-strand meshes. The results of this grid convergence study clearly show that for this test case one could employ either form of mesh generation and achieve results of comparable accuracy on a given mesh. Furthermore both methods converge at approximately the same asymptotic rate. The drag error in this case is take as the difference between the computed drag coefficient and the drag coefficient computed with the GGNS[26–28] flow solver developed at Boeing. The reference drag value can be found on the LaRC turbulence modeling webite.[19]



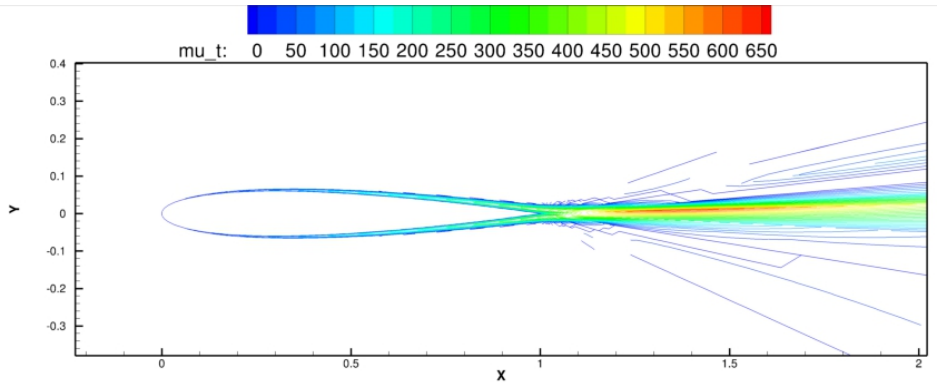
(a) Computed Mach number using a bent-strand mesh with $N = 16,898$ elements



(b) Computed Mach number using a multi-strand mesh with $N = 19,397$ elements



(c) Computed μ_T using a bent-strand mesh with $N = 16,898$ elements



(d) Computed μ_T using a multi-strand mesh with $N = 19,397$ elements

Figure 16. Flow solution contours for Mach number and turbulent eddy viscosity using bent and multi-strand meshes.

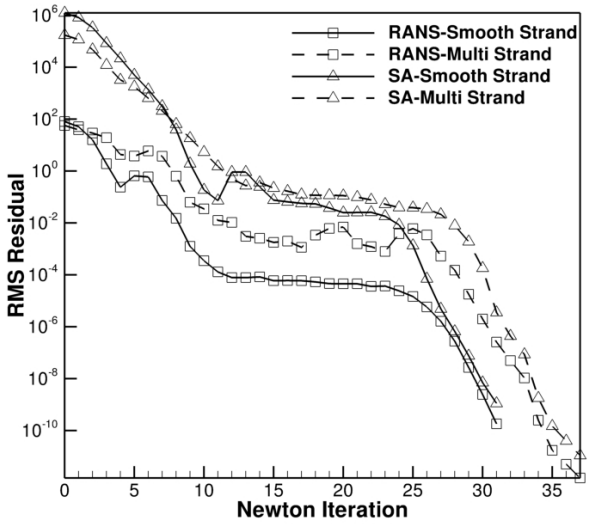


Figure 17. Convergence of the non-linear solution residuals for the finest bent- and multi-strand meshes.

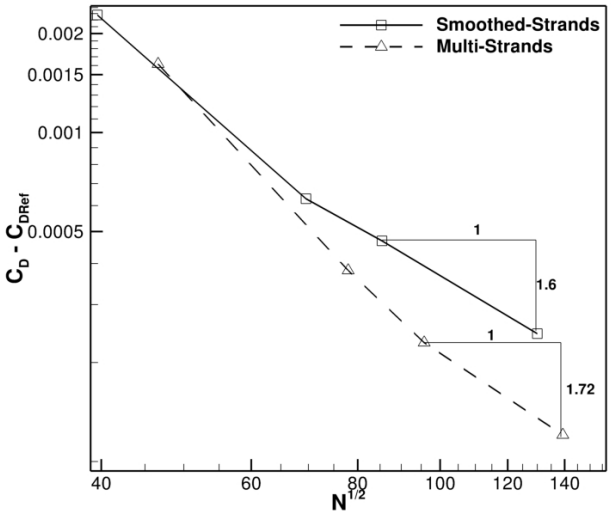


Figure 18. Grid Convergence of the computed drag coefficient for flow over a NACA0012 airfoil using a sequence of bent- and multi-strand meshes.

VII. Conclusions

This paper describes two advances in the development of automated viscous meshing capabilities using the Strand/Cartesian dual-mesh grid paradigm. The first is the addition of RANS-SA capability to the laminar solver presented in previous work. Validation of the new RANS-SA capability on flat plate and airfoil at various angles of attack reveal the strand solver is comparable in accuracy to other production-level RANS codes such as CFL3D and FUN3D. Parallel scaling reveals the code shows good weak scaling overall and good strong scaling as long as the problem size is larger than to 100K cells per processor.

The second new capability is a new strand mesh generation procedure that seeks to avoid poorly shaped prisms that arise at sharp convex corners and saddle points with our current strand mesh generation scheme. Inputs include a solid geometry representation, such as geometry from a modern CAD system, and a desired surface tessellation. From there, the scheme constructs a surface tessellation and emits multiple strands from identified geometric corners and edges. The entire scheme is fully automated, constructing a strand volume mesh directly from the surface at runtime without intervention by the user. Prototype high-Re 2D RANS-SA calculations over an airfoil reveal good results obtained on these automatically-constructed meshes. with multiple strands applied at the sharp trailing edge, are comparable in accuracy to calculations performed on traditional manually-generated unstructured meshes.

The results presented indicate that the strand paradigm can be a viable option automatic viscous mesh generation for RANS-SA calculations. However, the focus of this paper was on fundamental validations and parallel scaling analysis of components of the Strand/Cartesian mesh generation and solution process. Future work should validate the automatic mesh generation and solver on practical 3D geometrically-complex engineering applications.

Acknowledgments

Material presented in this paper is a product of the CREATE-AV Element of the Computational Research and Engineering for Acquisition Tools and Environments (CREATE) Program sponsored by the U.S. Department of Defense HPC Modernization Program Office. This work was conducted at the High Performance Computing Institute for Advanced Rotorcraft Modeling and Simulation (HIARMS).

References

[1]Aftosmis, M.J., Berger, M.J., Alonso, J.J., "Applications of a Cartesian mesh boundary-layer approach for complex configurations", AIAA 2006-0652, 44th AIAA Aerospace Sciences Meeting and Exhibit, Reno NV, Jan 2006.

[2]Murman, S.M., Aftosmis, M.J., and Nemec, M., "Automated parameter studies using a Cartesian method.", AIAA Paper 2004-5076, 22nd Applied Aerodynamics Conference, Providence RI, Aug 2004.

[3]Coirier, W.J., and K. G. Powell, "Solution-Adaptive Cartesian Cell Approach for Viscous and Inviscid Flows," *AIAA Journal*, Vol. 34, 1996, pp. 938-945.

[4]Nakahashi, K., "Building-Cube Method; A CFD Approach for Near-Future PetaFlops Computers," *5th. European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS 2008)*,June 30–July 5, 2008 Venice, Italy.

[5]Nemec, M., Aftosmis, M.J., and Wintzer, M., "Adjoint-Based Adaptive Mesh Refinement for Complex Geometries," AIAA-2008-0725, 46th AIAA Aerospace Sciences Meeting, Reno NV, Jan 2008.

[6]Nemec, M., and Aftosmis, M.J., "Parallel Adjoint Framework for Aerodynamic Shape Optimization of Component-Based Geometry," AIAA-2011-1249, 49th AIAA Aerospace Sciences Meeting, Orlando FL, Jan 2011.

[7]Meakin, R., A. M. Wissink, W. M. Chan, S. A. Pandya, and J. Sitaraman, "On Strand Grids for Complex Flows," AIAA-2007-3834, 18th AIAA Computational Fluid Dynamics Conference, Miami FL, June 2007.

[8]Wissink, A.M., A.J. Katz, W.M. Chan, R.L. Meakin, "Validation of the Strand Grid Approach," AIAA-2009-3792, 19th AIAA Computational Fluid Dynamics Conference, San Antonio TX, June 2009.

[9]Katz, A., A.M. Wissink, V. Sankaran, R.L. Meakin, W.M. Chan, "Application of Strand Meshes to Complex Aerodynamic Flowfields," *Journal of Computational Physics*, Vol. 230, No. 17, July 2011.

[10]Katz, A., A. Wissink, V. Sankaran, "Convergence Acceleration Techniques for Coupled Adaptive Cartesian-Strand Grid Solutions," AIAA-2011-772, 49th AIAA Aerospace Sciences Meeting, Orlando FL, Jan 2011.

[11]Katz, A.J., A.M. Wissink, "Efficient Solution Methods for Strand Grid Applications," 30th AIAA Applied Aerodynamics Conference, New Orleans LA, June 2012.

[12]Work, D., O. Tong, A. Katz, A. Wissink, "Strand Grid Solution Procedures for Sharp Corners," AIAA-2013-0800, 51st AIAA Aerospace Sciences Meeting, Grapevine TX, Jan 2013

[13]Wissink, A.M., B. Jayaraman, A. Datta, J. Sitaraman, M. Potsdam, S. Kamkar, D. Mavriplis, Z. Yang, R. Jain, J. Lim, R. Strawn, "Capability Enhancements in Version 3 of the Helios High-Fidelity Rotorcraft Simulation Code," AIAA-2012-0713, 50th AIAA Aerospace Sciences Meeting, Nashville TN, Jan 2012.

[14]Spalart, P.R., and S.R. Allmaras, "A one-equation turbulence model for Aerodynamic flows," *Le Recherche Aerospátiale*, Vol. 1, pp. 5–21, 1994.

[15]Katz, A., and V. Sankaran, "Discretization Methodology for High Aspect Ratio Prismatic Grids," AIAA-2011-3378, 20th AIAA Computational Fluid Dynamics Conference, Honolulu HI, June 2011.

[16]Roe, P.L., "Approximate Riemann Solvers, Parameter vectors, and Difference Schemes," *J. Comput. Phys.*, Vol. 43, pp. 357–372, 1981.

[17]Diskin, B., J. Thomas, E. Nielsen, and H. Nishikawa, "Comparison of Node-Centered and Cell-Centered Unstructured Finite-Volume Discretizations. Part 1: Viscous Fluxes," AIAA-2009-0597, 47th AIAA Aerospace Sciences Meeting, Orlando FL, Jan 2009.

[18]Allmaras, S.R., Forrester, J.T., and Spalart, P.R. "Modifications and Clarifications for the Implementation of the Spalart-Allmaras Turbulence Model", ICCFD-1902, 7th International Conference on Computational Fluid Dynamics (ICCFD7), Big Island, Hawaii, July 2012.

[19]National Aeronautics and Space Adiminstration. "Turbulence Modeling Resource", NASA Langley, 2012. http://turbmodels.larc.nasa.gov/.

[20]Ladson, C., "Effects of Independent Variation of Mach and Reynolds Numbers on the Low-Speed Aerodynamic Characteristics of the NACA 0012 Airfoil Section," *NASA TM 4074*, Oct 1988.

[21]Karypis, G., and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, Vol. 20, No. 1, pp. 359–392, 1999.

[22]Sitaraman, J., B. Roget, and A. Wissink, "OSCAR - An Overset Grid Assembler for Overlapping Strand/Cartesian Mesh Systems," 11th Symposium on Overset Composite Grids and Solution Technology, Dayton OH, Oct 2012. http://2012.oversetgridsymposium.org/.

[23]Burgess, N. K , "An Adaptive Discontinuous Galerkin Solver for Aerodynamic Flows", Ph.D. thesis, University of Wyoming, Novmeber 2011.

[24]Burgess, N.K. and Mavriplis, D.J. "Robust Computation of Turbulent flows using a Discontinuous Galerkin Method", AIAA-2012-457, 50th AIAA Aerospace Scieneces Meeting, Nashville TN, Jan 2012.

[25]Burgess, N.K. and Mavriplis, D.J. "High-order Discontinuous Galerkin Methods for Turbulent High-lift Flows", ICCFD-4202, 7th International Conference on Computational Fluid Dynamics (ICCFD7), Big Island, Hawaii, July 2012.

[26]Johnson, F. T., Kamenetsky, D. S., Melvin, R. G., Venkatakrishnan, V., Wigton, L. B., Young, D. P., Allmaras, S. R., Bussoletti, J. E., Hilmes, C. L., "Observations Regarding Algorithms Required for Robust CFD Codes", to appear in *Modern Trends in Computational Aerodynamics – a Special Thematic Issue of Mathematical Modeling of Natural Phenomena*, June 2011.

[27]Allmaras, S. R., Bussoletti, J. E., Hilmes, C. L., Johnson, F. T., Melvin, R. G., Tinoco, E. N., Venkatakrishnan, V., Young, D. P., "Algorithm Issues and Challenges Associated with the Development of Robust CFD Codes", *Variational Analysis and Aerospace Engineering, Springer Optimization and Its Applications*, 2009, Volume 33, pp. 1–19;*note: authors in alphabetical order; F. T. Johnson primary author*

[28]Venkatakrishnan, V., Allmaras, S. R., Kamenetskii, D., Johnson, F. T., "Higher Order Solutions for the Compressible Navier-Stokes Equations", AIAA Paper 2003-3987, June 2003.